(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 March 2002 (14.03.2002)

PCT

(10) International Publication Number
WO 02/21314 A2

(51) International Patent Classification[7]: G06F 17/00

(21) International Application Number: PCT/US01/27662

(22) International Filing Date:
6 September 2001 (06.09.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/658,416 8 September 2000 (08.09.2000) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 09/658,416 (CON)
Filed on 8 September 2000 (08.09.2000)

(71) Applicant *(for all designated States except US)*: ASERA, INC. [US/US]; 600 Clipper Drive, Suite 100, Belmont, CA 94002 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: NOIK, Emanuel, G. [CA/US]; 2318 Lass Drive, Santa Clara, CA 95054 (US).

(74) Agent: WHEELER, Jeffrey; McAndrews, Held, and Malloy, Ltd., 500 W. Madison, Suite 3400, Chicago, IL 60661 (US).

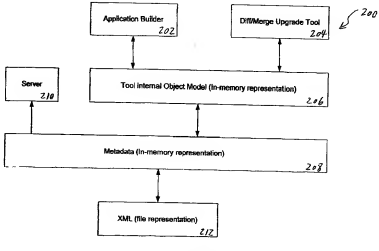
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: INTEGRATED DESIGN ENVIRONMENT FOR A COMMERCE SERVER SYSTEM



(57) Abstract: An integrated design environment enables the formation of business applications for usage on a commerce server system. The business applications consist of a variety of components including but not limited to business objects, business steps, behavior and display filters, and business rules arranged as a workflow. The design environment provides graphical interfaces for arrangement and modification of the components. The project based design environment includes project files with components stored as files, with the files often being functionally interdependent. The design environment provides for integrated arrangement of the components via the convenience of a graphical interface, and provides for validation of the interdependent components once arranged. A business application can thereby be viewed and/or developed with automatic crosschecking and generated documentation of the various interdependent component parts. This arrangement is facilitated, in part, through an interim metadata representation that encodes basic information for use by a semantically richer internal object model, and the overall commerce server.

WO 02/21314 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTEGRATED DESIGN ENVIRONMENT FOR A COMMERCE SERVER SYSTEM

RELATION TO PRIOR APPLICATIONS

This application is related to U.S. Provisional patent application having serial number 60/164,021, entitled "Method and Apparatus to Provide Custom Configurable Business Applications From a Standardized Set of Components," filed August 23, 1999, by one of the same inventors, and which is hereby incorporated by reference in its entirety. This application is also related to utility patent application having serial number 09/440, entitled "Method for Providing Custom Configurable Business Applications from a Standardized Set of Components," filed November 15, 1999, by one of the same inventors, and which is hereby incorporated by reference in its entirety. This application is also related to utility patent application having serial number 09/439,764, entitled "Apparatus to Provide Custom Configurable Business Applications from a Standardized Set of Components," filed November 15, 1999, by one of the same inventors, and which is hereby incorporated by reference in its entirety.

FIELD OF THE INVENTION

The present invention relates generally to an integrated design environment that provides various graphical tools and interfaces to facilitate the efficient formation and verification of business applications for running on a commerce server system.

BACKGROUND OF THE INVENTION

The prior referenced applications provide for methods and apparatuses for creating custom configurable business or channel applications from a standardized set of components. More specifically, the referenced invention allows each business to select from a set of applications, customize that set of applications,

and/or develop new customized applications from a set of development components. The prior applications provide for a server based method wherein best-of-breed services and/or applications are integrated in a seamless fashion and offered to enterprise businesses which develop customized business service applications through using the system. The server device is previously (and hereafter) referred to as the Asera Commerce Server (ACS).

The ACS includes a Commerce Server that provides a core set of technology (or application) services. A unique architecture and framework are provided by the Commerce Server which facilitates development and use of customized applications. All interactions with external systems or users are managed as business objects. The service application code is maintained separate from the data, thereby enabling the system to quickly include or change new business processes or technology components without having to write substantial amounts of new code. The business result is more rapid customer deployments and/or modifications that are customized to the level of including (if desired) the proprietary or competitive business practices of a contracting company.

The ACS can be viewed as a form of ASP (Application Service Provider). An ASP is generally an outsourcing business model. The ASP business model requires an open and extendable architecture that allows a system to implement a customer specific business solution in a short period of time. The ACS takes best-of-breed applications and incorporates them into one integrated solution to provide the ASPs. The architecture is scalable and extensible. A customized business (or channel) application solution is built for each enterprise company. The solution uses a "modular" or step-wise or "plug and play" approach towards building new applications. An enterprise company can then quickly acquire a turn-key e-commerce solution to automate their channel relationships. The system presents little (or no) risk for the enterprise company because a solution is built by the present system. The costs of undertaking such a development exist as a fixed development cost of the system. Any resulting customized solutions are implemented in considerably less time than previous systems. The enterprise

company might pay for the application services on a cost per transaction, or a fixed fee basis.

The ACS is used to capture the particularized (or specific) business processes for a given customer, and these business processes are converted into a set of customized applications. The ACS uses business steps and rules to construct the application. The objects are data representations. The steps are business operations with a defined set of input and output ports, with each port also having a defined set of parameters. The business rules are used to capture customer specific business practices. A unique tool that employs a graphical user interface (GUI), allows a developer to arrange various steps (or composite steps) into business processes, or workflows. The tool provides library catalogs of steps to be applied to the various objects. The connections between steps are also verified as correct. A graphical display of the business process is shown, and rules can thereafter be applied to provide further customization by conditionally tagging certain points. Hence, to create a business process (or application) for any given business, tools are provided which allow modules (or steps) to be plugged or dropped into the potential process. The steps can be moved, or the connections modified. An initial person-to-person (or other type of) interview with the business (or customer) can be used to produce the framework for arranging the steps according to the needs of that particular business (i.e. customized routines). The modular aspect of the present system allows this to be done -- and modifications made -- in a relatively quick fashion. For instance, if a process has been created, but the customer wants it to behave in two different manners, then certain rules can be applied to provide the desired results, depending on conditional triggers that can be associated with the underlying business objects.

In general, Internet transactions can be divided into two categories: 1) business-to-business transactions, and 2) business-to-consumer transactions. Most solutions to automate transactions have dealt with business-to-consumer interactions. As such, these interactions are much more straightforward than business-to-business transactions. In a business-to-consumer transaction, the merchant supplies a "storefront" or web site that offers products to any number of

diversified consumers who might wish to view this web site. The consumer then purchases a product via a selection and payment method, and the product is thereafter shipped to the consumer. On the other hand, when one business deals with another business, there is a much greater amount of business processes and customization in the transactions that occur.

The prior referenced applications describe a graphical user interface (GUI) tool that can be used to selectively arrange the business objects or steps into a working application. The implementation describes various screens for manipulating the business steps and their associated input ports and output ports. Business rules can also be applied to guide the flow of the assembled steps. The resulting application corresponds to application code (XML or otherwise) that is used by the commerce system to achieve the particular (customized) business processes desired by the user. Still further details of the integrated design environment (i.e., application builder) tool implementation, and various enhanced features, are provided in the present invention in order to further describe and clarify the tool.

SUMMARY OF THE INVENTION

The present invention provides a data source layer framework for conveying data between business applications and external data sources. The underlying commerce system is first summarized, and thereafter the data source layer framework of the present invention is summarized.

To achieve the foregoing, and in accordance with the purpose of the present invention, the system described includes, for example purposes, a server device referred to as the Asera Commerce Server (ACS). The ACS provides for the development and implementation of customized, automated, web-based business service applications. The ACS provides for a server based method wherein best-of-breed services and/or applications are integrated in a seamless fashion and offered to enterprise businesses that develop customized business service applications through using the system.

The ACS includes a Commerce Server that provides a core set of technology (or application) services. A unique architecture and framework are provided by the Commerce Server that facilitates the methods described herein. All interactions with external systems or users are managed as business objects. The service application code is maintained separate from the data, thereby enabling the system to quickly include or change new business processes or technology components without having to write substantial amounts of new code. The business result is more rapid customer deployments and/or modifications that are customized to the level of including (if desired) the proprietary or competitive business practices of a contracting company.

The ACS thus can be viewed as an ASP, or Application Service Provider. An ASP is generally an outsourcing business model. The ASP business model requires an open and extendable architecture that allows a system to implement a customer specific business solution in a short period of time. The ACS takes best-of-breed applications and incorporates them into one integrated solution to provide the ASPs. The architecture is scalable and extensible. A customized business (or channel) application solution is built for each enterprise company. The solution uses a "modular" or step-wise or "plug and play" approach towards building new applications. An enterprise company can then quickly acquire a turnkey e-commerce solution to automate their channel relationships. The present system presents little (or no) risk for the enterprise company because a solution is built by the present system. The costs of undertaking such a development exist as a fixed development cost of the present system. Any resulting customized solutions are implemented in considerably less time than previous systems. The enterprise company might pay for the application services on a cost per transaction, or a fixed fee basis.

The ACS is used to capture the particularized (or specific) business processes for a given customer, and these business processes are converted into a set of customized applications. The ACS uses business steps, and rules to construct the application. The objects are data representations. The steps are business operations with a defined set of input and output ports, with each port

also having a defined set of parameters. The business rules are used to capture customer specific business practices. A unique tool that employs a graphical user interface (GUI), allows a developer to arrange various steps (or composite steps) into business processes, or workflows. The tool provides library catalogs of steps to be applied to the various objects. The connections between steps are also verified as correct. A graphical display of the business process is shown, and rules can thereafter be applied to provide further customization by conditionally tagging certain points. Hence, to create a business process (or application) for any given business, tools are provided which allow modules (or steps) to be plugged or dropped into the potential process. The steps can be moved, or the connections modified. An initial person-to-person (or other type of) interview with the business (or customer) can be used to produce the framework for arranging the steps according to the needs of that particular business (i.e. customized routines). The modular aspect of the present system allows this to be done -- and modifications made – in a relatively quick fashion. For instance, if a process has been created, but the customer wants it to behave in two different manners, then certain rules can be applied to provide the desired results, depending on conditional triggers that can be associated with the underlying business objects.

The underlying commerce server provides certain functional components. Thereafter, various vertical applications are built (and/or customized) for the particular enterprise business. A suite of best-of-breed components and underlying applications are chosen to comprise, or run on top of, the commerce server. These components can be selected as the best technologies to provide certain desired efficiencies, cost reductions, and the like. The best-of-breed components are thereafter incorporated (or interfaced) with the business applications through APIs. A catalog of APIs can be provided for developers of applications. If any underlying best-of-breed components change, then the commerce server configuration can be changed without interfering with any user interfaces.

The integrated design environment (IDE) of the present invention adds the graphical ability to take an XML (or other) representation of an application and

present it in a structured, hierarchical way for formation of an application to run on the commerce server system. The particular IDE discussed in the present invention is also referred to the "application builder." The analysis and formation of XML files via line editors has proven to be too cumbersome, error-prone, and time consuming. The files representing the various business objects (or the like) can be formed and cross-checked for accuracy in their interdependencies. This is facilitated by having the commerce server interact with a metadata representation of each application as formed from the integrated design environment. The metadata layer is capable of reading an XML file and creating a "bare-bones" in-memory object representation of a particular component. The metadata format captures only the essential data that represents a particular instance of an application component. The tool internal object model, however, is used to convert each basic metadata object into a semantically richer representation. The richer representation thereby allows additional logic and structure that permits the integrated design environment to present semantically rich user interfaces which support a variety of complex features.

The integrated design environment uses project-based development. A project-based approach groups a set of components into logical groups or modules, wherein most components typically depend on other components.

The present application also performs semantically rich component validation. More than parsing and evaluating the syntax of an XML file, the present system validates the content of the XML files. The tool is capable of performing deep semantic component validation, both of individual components, and more significantly, across components.

An "Aseradoc" feature is also available which provides for component documentation. Developers are able to document components within the tool, even without entered comments being directly associated with the code. Any comments that might have been entered, however, are integrally incorporated into the final generated documentation. The user-supplied documentation is persisted

in the component's XML file, which is more likely to be maintained (and persisted) than externally supplied comments.

Accordingly, one aspect of the present invention includes an integrated design environment to facilitate formation of a business application having business objects which are maintained as independent files, the resulting business application to be used on a commerce server system, the integrated design environment comprising: an area for displaying a project tree hierarchy and selectively choosing a project file for graphical display; a detail viewing area for selectively displaying a structured view of the project file, or displaying a workflow view of the project file; a properties configuration area for displaying and selecting properties associated with a selected item object in the detail viewing area; a comments area for displaying and facilitating entry of comments associated with a selected item object in the detail viewing area; and a message area for displaying a variety of warning, error, and/or status messages associated with validation of the project file in relation to the various interdependencies of the object files comprising the business application.

Another aspect of the present invention includes a formation and validation tool for building a business application having business objects which are maintained as independent files, the resulting business application to be used on a commerce server system, the tool comprising: a file representation; a metadata layer for reading the file representation and creating a basic object representation of a particular component; an object model representation layer that uses the metadata layer to capture information necessary to perform cross-component project validation; an application builder device that uses the object model representation layer to form and display applications; and a commerce server device that uses the metadata layer to retrieve the information needed to validate and/or execute the business objects.

Another aspect of the present invention includes an integrated design environment for the formation of applications to be run on a commerce server system, the integrated design environment comprising: a plurality of objects

represented as files for forming the applications, whereby each of the files can be interdependent upon other files within the system; a graphical display for arrangement of the plurality of objects; and a validation routine for verifying the interdependencies as being functionally correct for the given object arrangement.

These and other aspects and advantages of the present invention will become apparent upon reading the following detailed descriptions and studying the various figures of the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention, together with further advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings in which:

Figure 1 shows a block diagram, according to one aspect of the present invention, which shows interdependencies between files in the system.

Figure 2 shows a block diagram, according to one aspect of the present invention, which shows the tool internal object model, and associated metadata layer for formation (and crosschecking) of an application.

Figure 3 shows a block diagram, according to one aspect of the present invention, which shows a high level architecture of the Tool Internal Object Model.

Figure 4 shows a block diagram, according to one aspect of the present invention, of the elements comprising a document tree node.

Figure 5 shows a block diagram, according to one aspect of the present invention, wherein the interfaces between operands are facilitated through object choosers and editor invokers.

Figure 6 shows a block diagram with details of the information stored in the metadata layer.

Figures 7-19 show various screen displays of the integrated design environment, which serve to explain certain features integrated throughout.

DETAILED DESCRIPTION OF THE INVENTION

The present invention offers a low risk alternative for companies that wish to quickly deploy an automated business application to their customers. In doing so, the company will interact with Asera personnel to capture various business processes to be implemented for the company. Thereafter, the company will have these business applications running on the Asera servers for use by both the customers and suppliers. Accordingly, an integrated design environment is needed for efficiently developing business applications from the component parts as defined by the system. The integrated design environment should provide (at least) certain graphical interface tools that allow for arrangement of the workflow steps, and verification of the step arrangements thereafter.

To create ACS business applications, or the like, application engineers first create several types of components that are encoded and persisted as XML (Extensible Markup Language) files, or the like. These components include, but are not limited to: (1) Business Objects; (2) Business Object Configurations; (3) Business Steps, including (a) simple steps (including normal, virtual, implementation), (b) interactive steps (normal, virtual, implementation), (c) composite steps (normal, virtual, implementation); (d) bridge steps, (e) transition steps; (4) Behavior Filters; (5) Display Filters; (6) Message Resource files; (7) Image Resource files; (8) Enumerated types; (9) Named Candidates; (10) Named Criteria. In the present embodiment (and for example purposes), the components might be persisted as XML files, or HTML files. In addition to being XML files (or HTML, or other languages) files, the objects, configurations, and steps can also refer to each other. Hence, any given file may not be independent.

Referring now to Figure 1, a block diagram 100 is shown of the interdependencies that might exist between the representative files. Such interdependencies might include hyperlinks or other dependencies. File1 (102) is

shown interdependent with File3 (106). File2 (104) is shown interdependent with File1 and File3. File3 is similarly shown interdependent with File1 and File2. These interdependencies are meant to be representative only, and other combinations are meant to be included with an ever-increasing complexity of file structures.

An application engineer, professional services engineer, or the like can create and edit XML files by using a text or XML editor. However, this practice is tedious, error-prone, and time-consuming. In addition, certain types of artifacts, such as composite business steps (which can be used to capture workflow logic) are most naturally described in a graphical (diagrammatic) manner. Furthermore, even relatively simple artifacts, such as business objects, can be relatively complex to describe in XML, but very simple to describe using a customized graphical user interface (GUI).

For example Appendix B provides an XML representation of a business object definition. This particular BOD is the calculator example used throughout this description for example purposes. The line-by-line code proves to be difficult to read and the workflow represented by the XML code is difficult to discern from the code. In contrast, Appendix C shows a view provided by the present invention of the same business object definition. The structure view chosen provides a hierarchical listing in the display window, as shown. Other views and screens can similarly show the workflow as an easy to discern graphical workflow.

The Application Builder (referred periodically throughout this description as the "tool") is an integrated development environment (IDE) for creating and manipulating ACS artifacts. The tool consists of a general extensible framework and a set of custom plug-in "editors" - with one for each type of artifact. The tool performs additional input validation when loading XML artifacts, thereby identifying errors very early in the design process. Furthermore, each custom editor, to a large extent, also prevents the user from creating ill-defined or invalid artifacts.

Certain features of the tool include, but are not limited to the following:

(1) The tool uses project-based development. Most components will typically depend on other components. With a project-based approach, sets of components are grouped into logical groups or modules.

(2) GUI productivity features, such as cut/copy/paste, mouse-click navigation, and persistent user preferences and settings.

(3) Semantically rich validation. While XML parsers can be used to validate the syntax of an XML file (i.e., whether the file is "well-formed"), they cannot be used to validate the content of the XML files. The tool is capable of performing deep semantic component validation - both of individual components, and more significantly, across components. Since most components depend on other components, this cross-component validation is extremely powerful. It can significantly reduce development effort and time by clearly identifying a broad variety of errors very early in the software design cycle.

(4) Integrated build utility. This feature enables the developer to automatically generate various derived files, such as JSP source files, Java class files, generate makefile dependencies, and to automatically remove such generated files. Furthermore, the developer can apply such actions to the entire project or selectively - to subsets of the project.

(5) A documentation feature -- referred to as "Aseradoc" -- which allows a developer to document components within the tool. Such documentation is persisted in the component's XML file, making it more likely that the documentation will be kept up-to-date as the component is modified. Furthermore, the tool can format the user-supplied documentation into a collection of HTML files that can be viewed by a standard web browser. Again, unlike independent static documents that are less likely to remain current as the software components are modified, the tool can be used to easily generate the most current documentation with a single mouse click, thereby facilitating iterative software development.

An example of a sample Aseradoc output file is shown in Appendix A. The file represents an HTML file that was generated for a business object definition, and in particular the calculator example of the present detailed description. An attribute summary is provided, along with details of each of the attributes. Relations might also be shown, though this example does not contain any.

Referring now to Figure 2, a block diagram 200 is shown of a high level architecture representing the present system. Graphical tools such as the Application Builder 202 and the Diff/Merge upgrade tool 204 are built on top of a generic Tool Internal Object Model (TIOM) 206 as shown. In order to read or write an XML (or the other type) of file 212, the application builder 202 uses functionality provided by the metadata layer 208 that it shares with the server 210 (i.e., Asera Commerce Server or ACS). The metadata format captures only the essential data that represents a particular instance of an application component. The TIOM 206, on the other hand, converts each such metadata object into a semantically richer representation. This model provides additional logic and structure that allows the Application Builder 202 to present semantically rich user interfaces which support a variety of complex features.

The GUI might offer any of a variety of features, including for instance intelligent structure views. Such views might support common user actions across all components. Such actions might include (but are not limited to): (a) cloning of a selected child; (b) moving a selected child up or down relative to its siblings; (c) removing all children from a parent; (d) removing a particular child from a parent; (e) expanding and/or collapsing all children, a portion of the children, etc.; and (f) keyboard-based cut/copy/paste. The GUI might further provide context-sensitive property sheets, toolbars, and popup menus. Comments and/or documentation windows (panes) might further be included.

Regarding the cut/copy/paste feature, unlike a text editor in which a user cuts/copies/pastes text, the building blocks of ACS components are highly varied. Each component has its own rules that govern where such building blocks can appear. The tool supports this heterogeneous target/source model. For example, a

step input port can be pasted only inside an "input port's" construct of a step, and only if that "input port's" construct does not already contain the maximum allowable input ports. Similarly, the tool will prevent users from cutting or removing certain constructs, as doing so would violate a rule associated with the parent container that states that the parent must contain some minimum number of child components. For example, a simple step must always contain exactly one input port and one output port. In this case, both the input and output port can be copied, but neither one can be removed or cut. Similarly, other input or output ports cannot be pasted in the input port node of a simple step.

The GUI might further provide for a browser-style forward/back selection history mechanism. Other navigation commands might include a double mouse-click as a means to drill down to details or to jump to related components. A persistent session history might also be provided. For instance, recent/last projects, window sizes and positions, and other session information is recorded and automatically maintained between sessions.

Referring now to Figure 3, a block diagram 300 is shown with further representative details of the Tool Internal Model (206) from Figure 2. This diagram is generally meant to serve as a single consistent representation that captures (among other things) certain information necessary to perform (a) cross-component project validation; (b) generate documentation, and (c) application of standard UI features across different types of components. Note that the boldface connector arrow 302 indicates that a transition from one element to another has many instances, whereas the open-faced arrow 304 indicates only one instance.

The first element is the project 306. A project is a collection of web application components. A project file is itself an XML file that encodes the names of the constituent components and other project settings. In memory, each project object contains a collection of references to document objects.

The next element 308 represents a document object. A document object is the tool layer's variant of a metadata object. Each document object contains a

reference to a corresponding DocumentUI object 310, and a reference to a DocumentTreeNode object 312 that is the root of the Document's hierarchical representation. Each document is described as a hierarchy that is constructed from variants of the DocumentTreeNode objects. Each such variant can be customized to serve in a variety of situations. The key aspect of this mode of construction is that all DocumentTreeNode objects support a common set of functionality.

The DocumentUI object 310 is used to define the set of graphical views that a particular type of component exposes to the UI. For example, a composite step component exposes two views in the UI: a structural view and a "workflow" view. A business object definition, for example, exposes three views: a structural view, an attributes (tabular) view, and a relations (tabular) view.

The DocumentTreeNode 312 represents a tree structure of child nodes. Each node object contains an ordered collection of zero or more child nodes. These nodes include an extensible collection of properties, including but not limited to: optional name; optional comment text (both as a single text string, or multiple tagged comments); optional caption text. Each node has a reference to a NodePolicy object 314 (wherein each policy is typically shared among many nodes). Each node further includes an optional reference to a PrototypeGroup object 316. In addition, each subclass (variant) of DocumentTreeNode can be customized in many ways, including but not limited to: custom logic to expose properties in a "property sheet"; custom component validation logic; custom documentation generation logic; custom clipboard manipulation logic (that determines what constructs can be cut/copy/pasted); custom logic to determine minimum/maximum number of children constraints; and custom unique key generation logic.

The NodePolicy object 314 is used to describe the structure and behavior of a particular class of DocumentTreeNode objects. Specifically, a NodePolicy can be used to capture the following information: whether the children of the node need to have unique names; whether the node has a prototypical child instance, and a determination of that instance. Each such prototype can be used to create

additional children in a "cookie-cutter" fashion; actions that should be used to generate a new child; the set of actions supported by the node; a determination of the set of validator objects that can be used to validate an instance of the node. Notably, each DocumentTreeNode variant can implement its own custom validation by extending standard behavior; A determination of the set of standard documentation objects that can be used to generate automatic documentation (i.e., an "Aseradoc") for this node.

The PrototypeGroup 316 is an object that contains a collection of DocumentTreeNode, each of which is a prototypical instance of some DocumentTreeNode object. This collection of prototypes is used to define a list of valid types of objects.

The DocumentTreeNode 312 of Figure 3 is further detailed in Figure 4 via a representative example 400, in this case a calculator application. The BOD (Business Object Definition) 402 is first shown having a name that uniquely identifies the application in the system, i.e., com.asera...calculator, or the like (wherein any of a variety of identifier strings might be included in-between the particular words shown). The BOD also includes the name of other BODs that are extended by it. In general, this will include knowledge that the name refers to a BOD, and a chooser/invoker device can be used to select the particular BODs. Each BOD 402 contains (at least) a collection of attributes 404, a collection of relations 406, and a collection of configurations 408.

The attributes 404 might further include operations, including but not limited to operations and constraints/policies. Example operations might include: "add attribute" (to create an attribute object, i.e., 410 or 412 below); "remove children" (i.e., remove all attributes); "expand"; or "collapse." Example constraints/policies might include naming of the children (i.e., each of the children has a unique name); sorting of the children by name; and limitations on the minimum or maximum number of children (i.e., ranging from zero to infinity).

A number of attribute objects 410, 412, and so forth, are shown as a subset of the attributes 404. Each attribute object might include operations such as "clone" or "remove." The attribute objects further include properties such as: object name; domain (via select list); enumeration flag (i.e., IsEnum, Boolean); cardinality (via select list); requirement flag (Boolean); and/or persistence flag (Boolean). "Object choosers" might also be used to select valid references to other objects (i.e., interactive steps).

Moreover, some properties will be used to capture knowledge related to object and file references. For example, an interactive step refers to both display filters and HTML wireframe files. A simple step refers to Java source files. Steps in general might refer to other contained steps. In each such instance, the use of smart choosers/invokers forces the user to select "valid values." Referring now to Figure 5, Object 1 (502) uses Object 2 (504), Java File 1 (506) and HTML File 2 (506). The object type information is encoded to know what type of object chooser/editor invoker to apply in light of certain properties.

Referring again to Figure 4, configurations 408 are shown. This consists of a list of one or more business object configuration (BOC) components. As similar to relations (406) above, the name of each BOC should correspond to a valid BOC component. Accordingly, an object chooser and editor invoker are used to select and validate BOC components.

Figure 6 shows further details of the metadata component 208 from Figure 2, in light of the above calculator application example. As stated above, the metadata layer is capable of reading an XML file and creating a "bare-bones" in-memory object representation of a particular component. The metadata format captures only the essential data that represents a particular instance of an application component. This bare-bones representation does not facilitate the validation between the various components, as only basic information is captured. The metadata is shown have a list of information 602 pertaining to each BOD. In this instance, the information pertaining to each attribute is listed, i.e. name = Operand1, type = integer, cardinality = single, IsRequired flag = false, IsPersistent

flag = true, and the name/list of a business object that this attribute extends. Similar attributes are also listed. Relations are shown, including (for example) name, BOD name, cardinality, and IsRequired. A list of components (not shown) might similarly be listed.

Figures 7-19 show certain screen displays of the example calculator application as formed using the present system. The tool herein might be executed as a stand-alone Java application. Alternatively, the tool might also run as a Java applet that is executed within a web browser. As indicated above in the high-level figures, the application builder operates on projects. A project file contains a list of business objects, configurations, business steps, and wire frame files that are contained in the project. Project files are readily created, added, or removed via normal computer operations.

Figure 7 shows the main window 700 of the Application Builder, which consists of several areas. The project tree area 702 displays a folder hierarchy of project items. The detail pane 704 contains a set of tabbed panes that display detailed information about the selected project item. In this example, the detail pane presents a detailed view of a composite business step. The message pane 706 displays a variety of warning, error, and status messages. For instance, double-clicking on an error message will drive the selection in the detail view to display the error item. The property sheet 708 displays properties for the selected item in the detail pane. The comments pane 710 displays user-entered comments associated with the selected item in the detail pane. The context-sensitive toolbar 712 displays actions that can be performed to the selected item in the detail pane. This same set of actions can be accessed by right-clicking on the item in the detail pane.

The tool's GUI is similar to other Integrated Design Environments (IDEs). As such, the top left pane displays a tree of project items. Any item can be displayed and edited by double-clicking on its name in the project tree control or right clicking on its name and selecting "edit" from the popup menu. The item details are displayed in a tab control located in the top right pane. The tab control

contains several tabs, with the set of tabs that is displayed depending upon the type of item being edited. Modified files might be saved by right-clicking on an item to be saved (in the project tree), and selecting "save" from the popup menu, or invoking "file-save" or "file-save-all" from the toolbar.

Referring now to Figure 8, each business object definition (BOD) 800 has the following tabs: A structure view 802, which is a tree view of the attributes and relations of a business object; Attributes 804, wherein this tab is used to manipulate the attributes of a business object. Each attribute has properties such as name, domain (data type), cardinality, and flags (i.e., a flag to indicate if the attribute is used as a key, or if the attribute is required, or if the attribute is persisted). Relations 806, wherein this tab is used to manipulate the relations of a business object. Each relation has properties such as name, type (Business Object), cardinality, required and persisted flags, and association which can have a value of "reference" (i.e., the object contains a reference to a business object of the specified type), or "containment" (i.e., the object contains an instance of the business object of the specified type).

Each node in the structure view 802 can be manipulated by either invoking an action or by modifying a property. When a tree item 808 is selected, the toolbar 814 is automatically updated to show actions that can be applied to the selected item. Similarly, the property sheet 812 automatically displays properties associated with the selected item. Actions are invoked by clicking one of the toolbar buttons 814, or by selecting the action from a context-sensitive popup menu 810. Properties are modified in the property sheet 812.

Figure 8A further shows a screen 850 with the attributes 804 tab selected. This screen provides a tabular (spreadsheet) view of the attributes of the BOD. This spreadsheet provides an overview of the attributes and their key properties, i.e., name, domain, cardinality, and relevant flags (required and persistent).

Figure 9 shows the business object configuration represented as a tree 900 within the GUI. The tree contains several top-level folders, i.e., (1) In-UseItems;

(2) Unused Items; (3) Name Groups; (4) Named Rules, wherein users can create one or more name rules. Rules are made up of a set of actions and conditions; (5) Object Triggers, wherein users can associate a number of rules with object-level trigger points, as is similarly done for individual attributes and relations. Such trigger points might include "BeforeFetch", "AfterFetch", "BeforeChange", "AfterChange", "AfterCreate", and "BeforeDelete"; (6) Queries; (7) Properties.

Figure 10 further shows a screen 1000 for the manipulation of business steps. Business steps are edited using several types of detailed views. For instance, all business steps contain the "structured view" tabs 1002 that can be used to display, create, delete, and modify input and output ports, and a set a variety of top-level properties. Composite steps are thereafter defined in the "work flow" tab 1004, which implements a graphical drawing (or workflow) editor, as presently shown in the figure. By default, the workflow editor can be used to move step instances by clicking the left mouse button on a step instance label, dragging the label, and the placing the step instance in the desired location (i.e., drag-and-drop operation). The default popup menu can be displayed by clicking the right mouse button on an empty space. To create a new step instance, the user can select "add step instance" from the default popup menu. To create a new link, the user can select "add link" from the default popup menu, click on the source step instance, drag the item to the target step instance, and place it thereafter.

Step instances and links can be edited by clicking on the instance of the label or link selection rectangle, and selecting the desired editing command from the step instance or link popup menus. When a step instance or link is first created, it is drawn in a light (gray or otherwise) color, thereby indicating that the step instance has not been mapped to a step definition, or the link has not been mapped to the input and output ports of its target and source step instances. When a step or link is correctly mapped, then the associated step instance label or link selection rectangle is drawn in another color (i.e., green). The instance label or link selection rectangle is drawn in yellow or red to indicate that one or more of the ports of the step instances are invalid or one or more of the transitions represented by the link are invalid. In such a case, the user should right-click on the invalid

object to bring up a context menu that will indicate which port or transition is invalid. Selecting the submenu associated with this port or link will allow the user to select a "describe error" action from the menu. This will in turn display a descriptive error message in the messages pane.

Links may represent two or more transitions, in which case the link selection rectangle will be replaced by two stacked rectangles. The popup menu for multiple transition links will list all transitions between the pair of step instances. The editor automatically generates several step instances, i.e., "enter", "exit", and "error" which are used to stitch incoming and outgoing port parameters into/out of the composite step, and to handle runtime errors, respectively.

In order to map, or "stitch" port parameters, a user should right-click on a link selection rectangle and select "map parameters..." from the appropriate link pull-right menu. The tool thereby allows users to stitch outgoing port parameters, local step variables, and global application variables into and out of step instances.

In this particular example, the workflow view 1006 shows a link 1008 displayed in a warning color (such as red), indicating that the transition is invalid. Popping up the context menu on this link reveals that the "oport:iport" arc is invalid 1010. Selecting the "describe error" command from the arc's context menu 1012 displays the errors described in the messages pane 1014. To validate the entire project, the user can click on the "validate" button, or select "tools-validate project" from the toolbar 1016. Again, this will generate a series of error, warning, and informational messages to be displayed in the messages pane 1014. Double-clicking on any message will cause the tool to select the relevant object and display the error location.

Behavior filters can next be manipulated, as shown by the screen 1100 in Figure 11. Behavior filters capture what information should be displayed in a region of a dynamically generated HTML page. For example, the above filter 1102 states that the user should be able to enter a text value for "operand1". However, this filter does not state how the text entry will be performed (e.g., using

a text field versus a text area HTML control). As a second example, "operator" 1104 is defined as a "singleselect" behavior, meaning that the user has the ability to select one value out of a list of possible values. Again however, this does not specify how the user will choose the value (i.e., checkboxes, drop-down lists, etc.).

Display filters are next shown being manipulated via screen 1200 in Figure 12. In this example display filter, the user can arrange the elements that correspond to the behavior elements defined in the behavior filter into an arrangement of rows, columns, blocks, etc. For each display element, the user can select one of the valid ways to "implement" the associated behavior. For example, the "operand1" text entry behavior is implemented as a single-line text input field 1202. The user can then select an alternative display style from the list of valid display styles for the given behavior element via 1204.

Screen 1300 in Figure 13 next shows how each display filter can also be viewed graphically. In other words, a graphical representation is shown that approximates the way the JSP (Java Script program) generated from this display filter would be rendered in the HTML browser. As the user modifies the structure or display elements of the display filter in the structure view 1302, the form view (as shown) 1304 automatically updates to show the revised structure/content.

Figure 13A further shows a construction screen for use in laying out the actual appearance of the displayed elements or components. The palette 1350 provides layout options of columns, rows, or blocks. A scrollable window 1352 provides a list of objects that can be placed for display in the various layout options. The results window 1354 allows for arrangement and display of the blocks, rows, and columns into a working display 1356. Through drag-and-drop operations (or the like), Operand 1 (1358) has been placed for display in the column entry 1360. Other objects can similarly be placed through the working display.

Figure 14 next shows an example page 1400 of the generated JSP showing the HTML output for the above display filter. Accordingly, the application

provides for the entry of a first operand 1402, a second operand 1404, and an operator 1406. By clicking on the "calculate" button 1408, a calculation result is produced.

Figure 15 next shows a screen 1500 with the integrated tool and utilities 1502 revealed in a pulldown menu. These tools and utilities include (but are not limited to): generate project makefiles -- makefiles are used to by the build utility to capture the list of target objects and dependencies; build project -- generates all files required by the project; clean project -- ; generate folder makefiles -- generates makefiles for selected folder and its children; build folder -- generates files that depend on objects in selected folder; build folder recursive -- generates files that depend on objects in selected folder and its children; generate folder dependencies -- generate dependencies for selected project; and clean folder -- remove generated files based on objects in selected folder and its children.

Figure 16 shows a screen 1600 resulting from the "validate project" selection 1602 from the tool menu. By invoking this command, a list of messages is generated in the message pane 1604. Double-clicking on the message in pane 1604 automatically selects the object and error location 1606.

Figure 17 next shows a screen 1700 resulting from selecting the "options" item from the tool pulldown menu. This screen allows the user to select the pathname for a Java source editor, which is used to edit Java source files. The screen also allows selection of an HTML viewer, which is used to edit HTML wireframe files.

Figure 18 next shows a screen 1800 with an example of defining object navigation via object choosers and invokers. Note that each microtemplate refers to a dynamically generated region of an HTML page. A display filter is used to describe the dynamically generated content. In this present example, the content of the microtemplate named "MT1" 1802 is generated by display filter "com.asera...dCalculator_Form". In order to select a valid value for a display filter, the user would click on the "..." (choose) button 1806 to launch a smart

object chooser (see Figure 19) that will display a list of valid display filters. In order to view or edit the corresponding display filter, the user would use an invoker by clicking on the "edit" button 1808. This will automatically select the display filter in the details pane. On the other hand, for wireframe candidates 1810, the chooser would show the list of available HTML files, while the invoker would launch the preferred HTML browser/editor of the user to view/modify the HTML file.

Figure 19 thereafter shows a screen 1900 which implements a smart object chooser that is used to select a valid display filter.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. Therefore, the described embodiments should be taken as illustrative and not restrictive, and the invention should not be limited to the details given herein but should be defined by the following claims and their full scope of equivalents.

CLAIMS

What is claimed is:

1. An integrated design environment to facilitate formation of a business application having business objects, business steps, and other components which are maintained as independent files, the resulting business application to be used on a commerce server system, the integrated design environment comprising:

an area for displaying a project tree hierarchy and selectively choosing a project file for graphical display;

a detail viewing area for selectively displaying structured and graphical views of the project file;

a properties configuration area for displaying and editing properties associated with a selected item in the detail viewing area;

a comments area for displaying and facilitating entry of comments associated with a selected item in the detail viewing area; and

a message area for displaying a variety of warning, error, and/or status messages associated with validation of the project file in relation to the various interdependencies of the object files comprising the business application.

2. The integrated design environment of Claim 1, wherein the detail viewing area displays various graphical views of particular project file when that project file is selected.

3. The integrated design environment of Claim 2, wherein a context-sensitive toolbar is provided with selectable actions dependant upon the items being selected in the detail viewing area.

4. The integrated design environment of Claim 3, wherein each defined business object includes a tabular view that shows a list of attributes that comprise the business object, and properties associated with each attribute.

5. The integrated design environment of Claim 4, wherein each defined business object includes a tabular view that shows a list of relations with other business objects, if any such relations exist.

6. The integrated design environment of Claim 1, wherein each composite business step includes a graphical workflow view that provides a graphical workflow editor to edit the step instances and links in the composite step.

7. The integrated design environment of Claim 6, wherein the workflow editor is used to move the step instances through drag-and-drop operations on labels associated with the step instances.

8. The integrated design environment of Claim 7, wherein the workflow editor is used to create and/or move links between source and target step instances.

9. The integrated design environment of Claim 8, wherein the source and target step instances have associated input and output ports, and the link instances are mapped to the input and output ports.

10. The integrated design environment of Claim 9, wherein when a step or link is correctly mapped, then the associated label is color coded to indicate its correctness, and when the step or link is incorrectly mapped, then the associated label is color coded to indicate its incorrectness.

11. The integrated design environment of Claim 10, wherein the incorrectly mapped step or link can be selected, and descriptive error messages will be shown in the message area.

12. The integrated design environment of Claim 1, wherein a behavior filter is used for arrangement of behavior elements, the behavior filter being used to capture information that should be displayed in a region of a dynamically generated display page.

13. The integrated design environment of Claim 12, wherein a display filter is used for arrangement of display elements that correspond to the behavior elements defined in the behavior filter, the display filter being used to arrange valid ways to implement the associated behavior.

14. The integrated design environment of Claim 13, wherein valid ways to arrange display elements include rows, columns, and tables.

15. A formation and validation tool for building a business application having components which are maintained as independent files, the resulting business application to be used on a commerce server system, the tool comprising:

a file representation;

a metadata layer for reading the file representation and creating a basic object representation of a particular component;

an object model representation layer that uses the metadata layer to capture information necessary to perform cross-component project validation;

an application builder device that uses the object model representation layer to form and display applications; and

a commerce server device that uses the metadata layer to retrieve the information from components needed to execute the business application.

16. The formation and validation tool of Claim 15, wherein the file representation includes a project file that encodes the names of constituent file components and other project settings.

17. The formation and validation tool of Claim 16, wherein the project file is written in a type of markup language.

18. The formation and validation tool of Claim 15, wherein the object model representation layer is semantically richer than the metadata layer.

19. The formation and validation tool of Claim 15, wherein the object model representation layer includes at least one project object, each containing a collection of references to document objects.

20. The formation and validation tool of Claim 15, wherein each document object reference contains a reference to a corresponding document user interface object, and a reference to a document tree node object that is the root of the hierarchical representation of the document.

21. The formation and validation tool of Claim 20, wherein each document is described as a hierarchy that is constructed from variants of the document tree node objects.

22. The formation and validation tool of Claim 21, wherein each variant can be customized to serve in a variety of situations.

23. The formation and validation tool of Claim 21, wherein the document tree node objects support a common set of functionality.

24. An integrated design environment for the formation of applications to be run on a commerce server system, the integrated design environment comprising:

a plurality of objects represented as files for forming the applications, whereby each of the files can be interdependent upon other files within the system;

a graphical display for arrangement of the plurality of objects; and

a validation routine for verifying the interdependencies as being functionally correct for the given object arrangement.

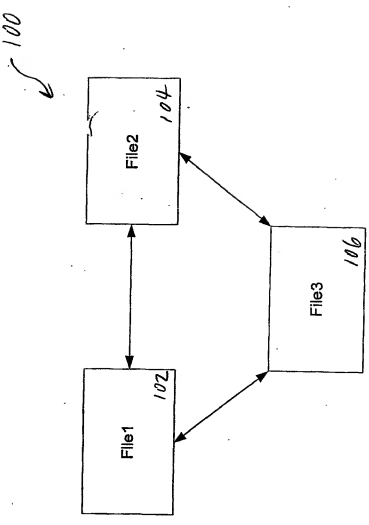


Figure 1

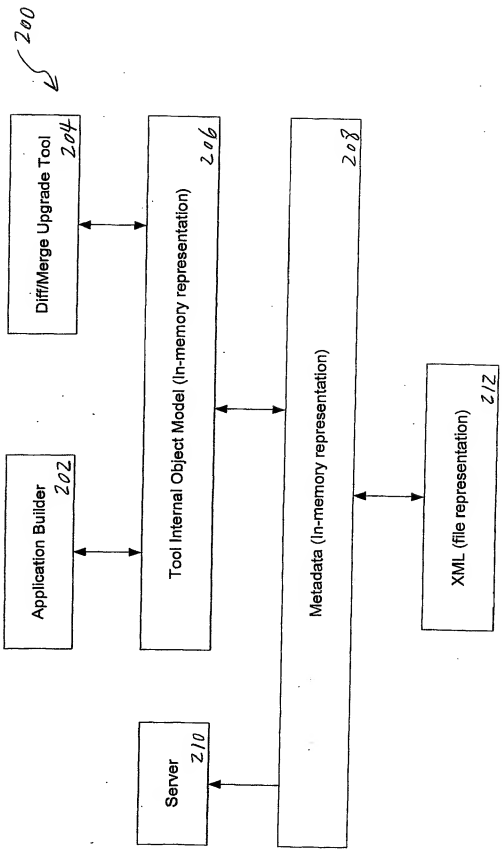

200
Diff/Merge Upgrade Tool
204
Application Builder
202
Tool Internal Object Model (In-memory representation)
206
Metadata (In-memory representation)
208
Server
210
XML (file representation)
212

Figure 2

3/21

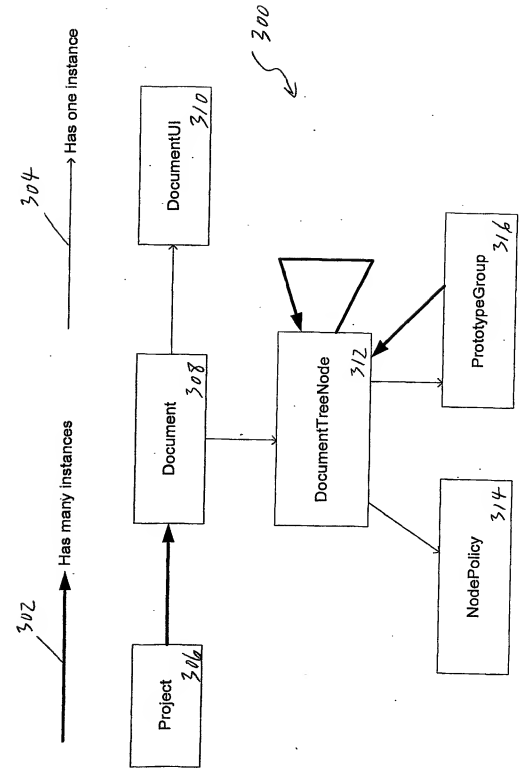

300
302
Has many instances
304
Has one instance
Project
306
Document
308
DocumentUI
310
DocumentTreeNode
312
NodePolicy
314
PrototypeGroup
316

Figure 3

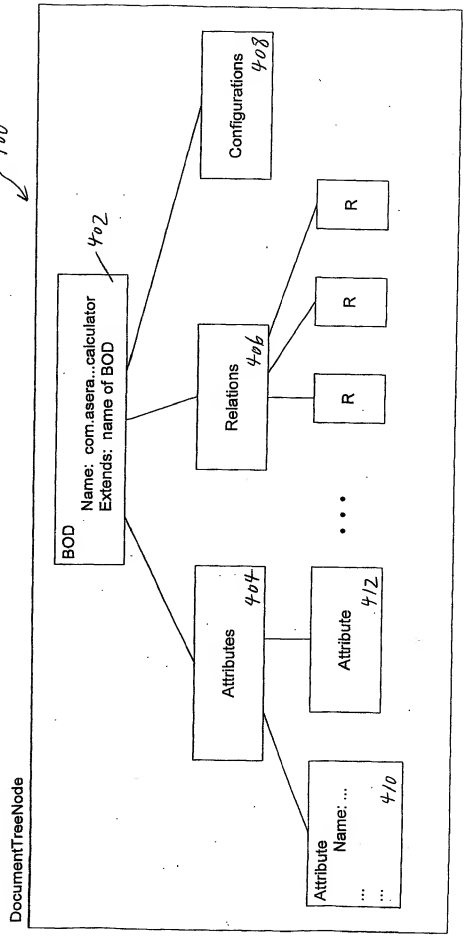

400
DocumentTreeNode
BOD
Name: com.asera..calculator
Extends: name of BOD
402
Configurations
408
Relations
406
R
R
R
Attributes
404
Attribute
412
Attribute
Name: ...
...
...
410

Figure 4

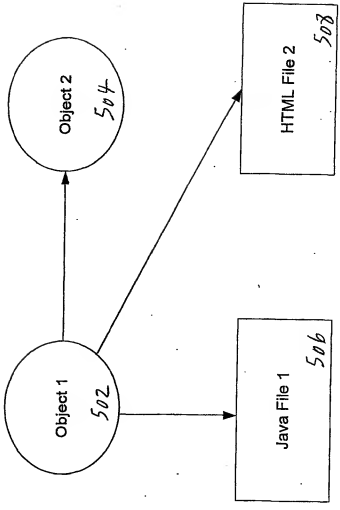


Figure 5

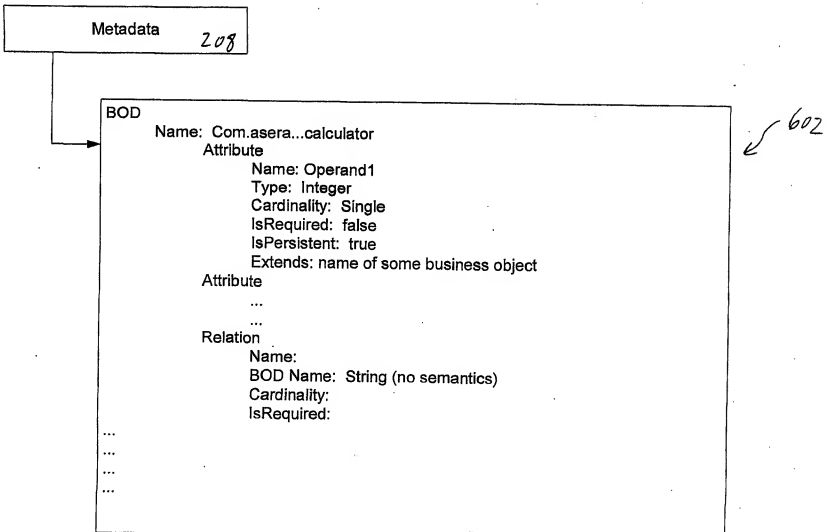

Metadata
208
BOD
Name: Com.asera...calculator
Attribute
Name: Operand1
Type: Integer
Cardinality: Single
IsRequired: false
IsPersistent: true
Extends: name of some business object
Attribute
...
...
Relation
Name:
BOD Name: String (no semantics)
Cardinality:
IsRequired:
...
...
...
...
602

Figure 6

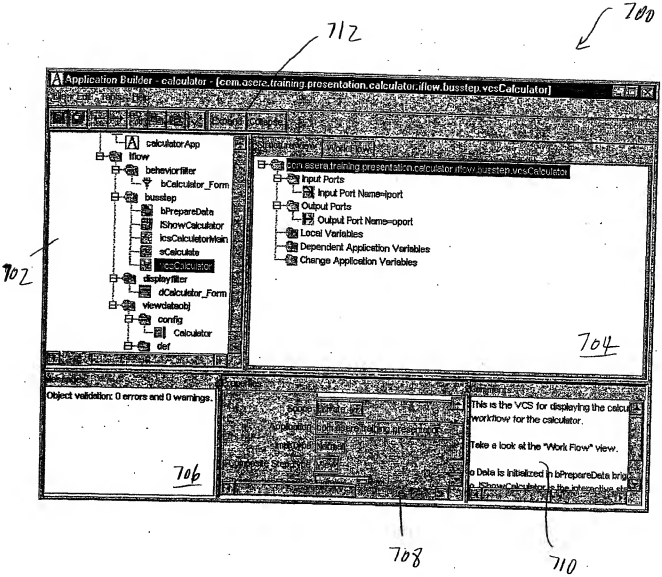

700
712
Application Builder - calculator - [com.acere.training.presentation.calculator.iflow.busstep.vcsCalculator]
calculatorApp
iflow
behaviorfilter
bCalculator_Form
busstep
bPrepareData
fShowCalculator
icsCalculatorMain
sCalculate
vcsCalculator
displayfilter
dCalculator_Form
viewdataobj
config
Calculator
def
702
com.acere.training.presentation.calculator.iflow.busstep.vcsCalculator
Input Ports
Input Port Name=iport
Output Ports
Output Port Name=oport
Local Variables
Dependent Application Variables
Change Application Variables
704
Object validation: 0 errors and 0 warnings.
706
This is the VCS for displaying the calcul
workflow for the calculator.
Take a look at the "Work Flow" view.
Data is initialized in bPrepareData brig
708
710

Fig. 7

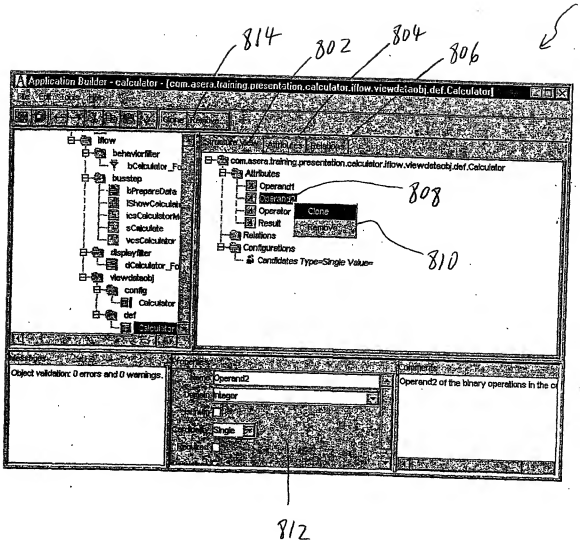

800
814
802
804
806
Application Builder - calculator - [com.asera.training.presentation.calculator.iflow.viewdataobj.def.Calculator]
iflow
behaviorfilter
bCalculator_Fc
busstep
bPrepareData
IShowCalculate
icsCalculatorIte
sCalculate
vcsCalculator
displayfilter
dCalculator_Fc
viewdataobj
config
Calculator
def
Calculator
com.asera.training.presentation.calculator.iflow.viewdataobj.def.Calculator
Attributes
Operand1
Operand2
Operator
Result
Relations
Configurations
Candidates Type=Single Value=
Clone
808
810
Object validation: 0 errors and 0 warnings.
Operand2
Integer
Single
Operand2 of the binary operations in the cl
812

Fig. 8

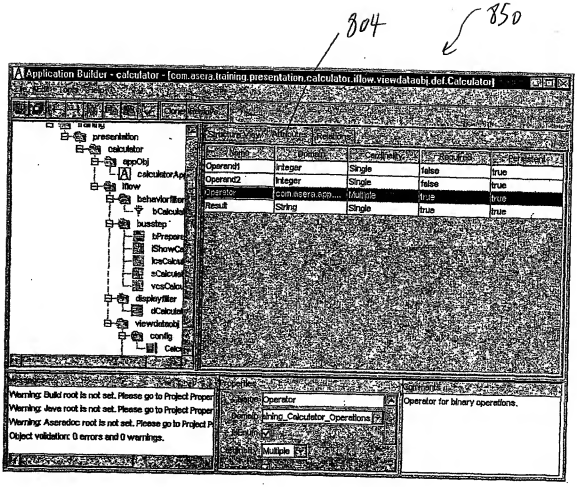

804
850
Application Builder - calculator - [com.asera.training.presentation.calculator.iflow.viewdataobj.def.Calculator]
presentation
calculator
appObj
calculatorApp
iflow
behaviorfilter
bCalculat
busstep
bPrepare
IShowCa
bcsCalcul
sCalculat
vcsCalcu
deployfilter
dCalculat
viewdataobj
config
Calc
Operand1 Integer Single false true
Operand2 Integer Single false true
Operator com.asera.app.... Multiple true true
Result String Single true true
Warning: Build root is not set. Please go to Project Proper
Warning: Java root is not set. Please go to Project Proper
Warning: Aseradoc root is not set. Please go to Project P
Object validation: 0 errors and 0 warnings.
Operator
ining_Calculator_Operations
Multiple
Operator for binary operations.

Fig. 8A

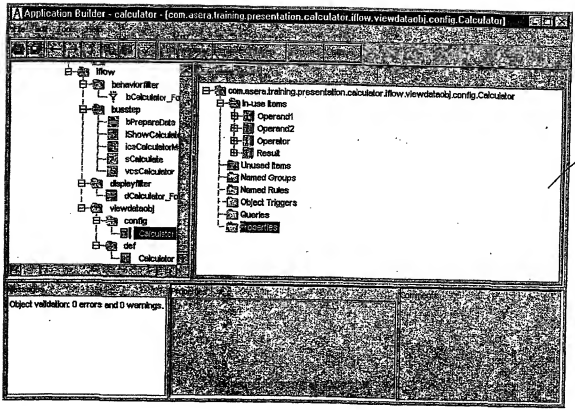

Application Builder - calculator - [com.asera.training.presentation.calculator.jflow.viewdataobj.config.Calculator]
jflow
behaviorfilter
bCalculator_Fc
busstep
bPrepareData
bShowCalculate
bsCalculateM
sCalculate
vcsCalculator
displayfilter
dCalculator_Fc
viewdataobj
config
Calculator
def
Calculator
com.asera.training.presentation.calculator.jflow.viewdataobj.config.Calculator
In-use Items
Operand1
Operand2
Operator
Result
Unused Items
Named Groups
Named Rules
Object Triggers
Queries
Properties
Object validation: 0 errors and 0 warnings.
900

Fig. 9

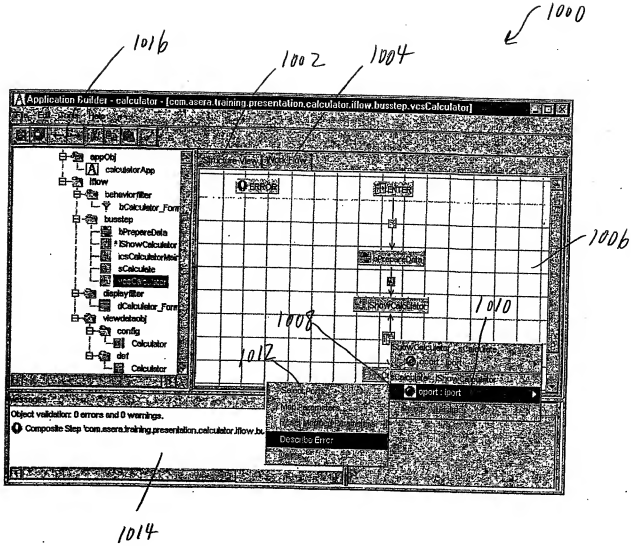

1000
1016
1002
1004
Application Builder - calculator - [com.asera.training.presentation.calculator.jflow.busstep.vcsCalculator]
appObj
calculatorApp
jflow
behaviorfilter
bCalculator_Form
busstep
bPrepareData
bShowCalculator
bcsCalculatorView
sCalculate
vcsCalculator
displayfilter
dCalculator_Form
viewdatalobj
config
Calculator
def
Calculator
ERROR
1006
1008
1010
1012
Object validation: 0 errors and 0 warnings.
Composite Step 'com.asera.training.presentation.calculator.jflow.bu
Describe Error
1014

Fig. 10

1100

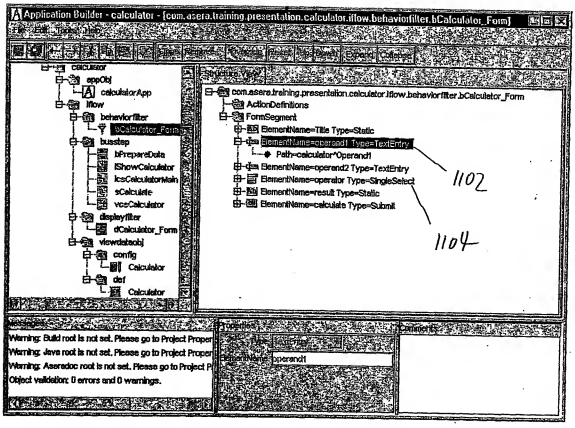

Application Builder - calculator - [com.asera.training.presentation.calculator.iflow.behaviorfilter.bCalculator_Form]
calculator
appObj
calculatorApp
iflow
behaviorfilter
bCalculator_Form
busstep
bPrepareData
bShowCalculator
bcsCalculatorMain
sCalculate
vcsCalculator
displayfilter
dCalculator_Form
viewdatobj
config
Calculator
def
Calculator
com.asera.training.presentation.calculator.iflow.behaviorfilter.bCalculator_Form
ActionDefinitions
FormSegment
ElementName=Title Type=Static
ElementName=operand1 Type=TextEntry
Path=calculator*Operand1
ElementName=operand2 Type=TextEntry
ElementName=operator Type=SingleSelect
ElementName=result Type=Static
ElementName=calculate Type=Submit
1102
1104
Warning: Build root is not set. Please go to Project Proper
Warning: Java root is not set. Please go to Project Proper
Warning: Asteradoc root is not set. Please go to Project P
Object validation: 0 errors and 0 warnings.
operand1

Fig. 11

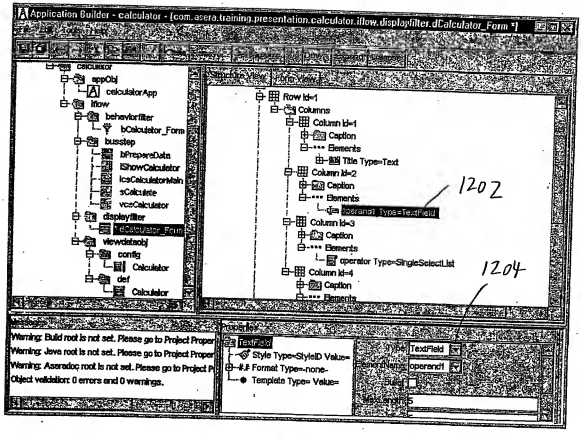

1200
Application Builder - calculator - [com.acera.training.presentation.calculator.iflow.displayfilter.dCalculator_Form *]
calculator
appObj
calculatorApp
iflow
behaviorfilter
bCalculator_Form
busobj
bPrepareData
bShowCalculator
bcsCalculatorMain
sCalculate
vcsCalculator
displayfilter
dCalculator_Form
viewdataobj
config
Calculator
def
Calculator
Row Id=1
Columns
Column Id=1
Caption
Elements
Title Type=Text
Column Id=2
Caption
Elements
operand Type=TextField
1202
Column Id=3
Caption
Elements
operator Type=SingleSelectList
Column Id=4
Caption
Elements
1204
Warning: Build root is not set. Please go to Project Proper
Warning: Java root is not set. Please go to Project Proper
Warning: Asseadoc root is not set. Please go to Project P
Object validator: 0 errors and 0 warnings.
TextField
Style Type=StyleID Value=
Format Type=-none-
Template Type= Value=
TextField
operand1
5

Fig. 12

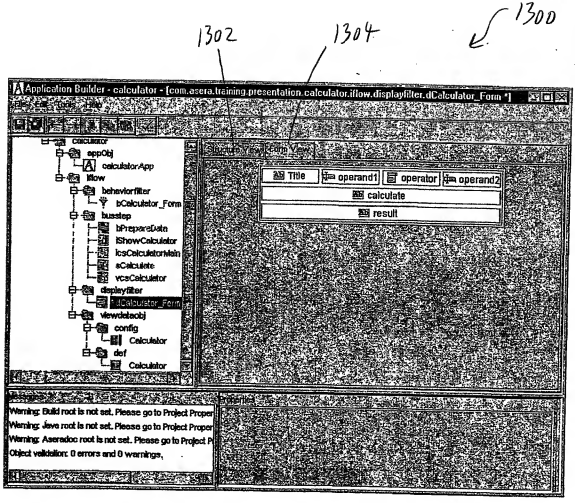

1300
1302
1304
Application Builder - calculator - [com.asera.training.presentation.calculator.iflow.displayfilter.dCalculator_Form *]
Standard View
Form View
Title
operand1
operator
operand2
calculate
result
calculator
appObj
calculatorApp
iflow
behaviorfilter
bCalculator_Form
busstop
bPrepareData
bShowCalculator
bsCalculatorMain
sCalculate
vcsCalculator
displayfilter
dCalculator_Form
viewdataobj
config
Calculator
def
Calculator
Warning: Build root is not set. Please go to Project Proper
Warning: Java root is not set. Please go to Project Proper
Warning: Aseradoc root is not set. Please go to Project P
Object validation: 0 errors and 0 warnings.

Fig. 13

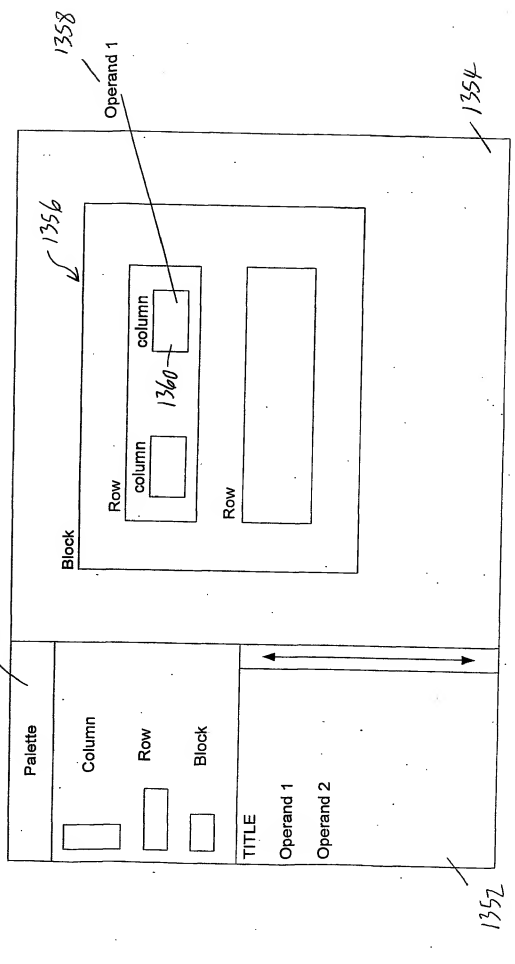

1350
Palette
Column
Row
Block
TITLE
Operand 1
Operand 2
1352
1354
1356
Block
Row
column
column
1360
Row
1358
Operand 1

Figure 13A

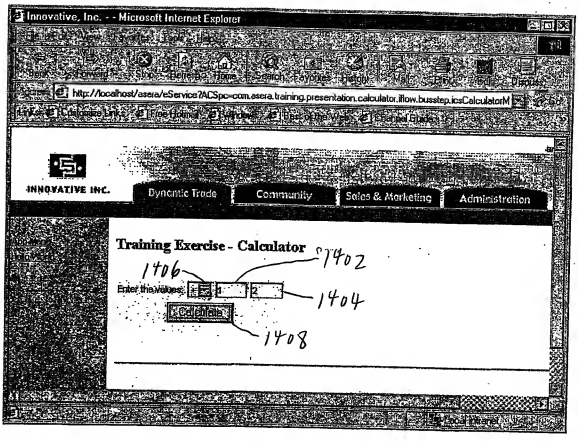

1400
Innovative, Inc. - - Microsoft Internet Explorer
http://localhost/asera/eService?ACSpc=com.asera.training.presentation.calculator.iflow.busstepicsCalculatorM
INNOVATIVE INC.
Dynamic Trade
Community
Sales & Marketing
Administration
Training Exercise - Calculator
1402
1406
Enter the values:
1404
Calculate
1408

Fig. 14

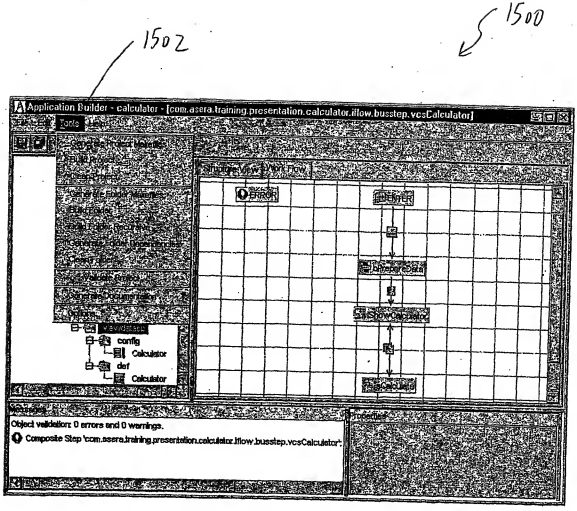
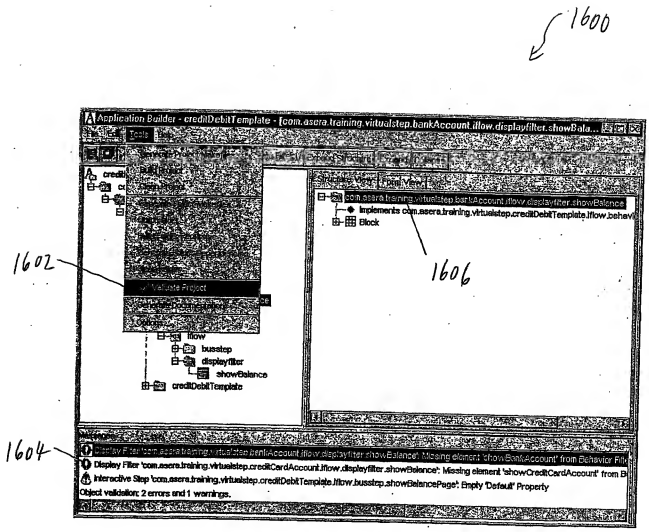

1502
1500
Application Builder - calculator - [com.asera.training.presentation.calculator.flow.busstep.vcsCalculator]
Tools
Begin
End
config
Calculator
def
Calculator
Object validation: 0 errors and 0 warnings.
Composite Step 'com.asera.training.presentation.calculator.flow.busstep.vcsCalculator'.

Fig. 15


1600
1602
1604
1606
Application Builder - creditDebitTemplate - [com.asera.training.virtualstep.bankAccount.iflow.displayfilter.showBala...
Validate Project
iflow
busstep
displayfilter
showBalance
creditDebitTemplate
com.asera.training.virtualstep.bankAccount.iflow.displayfilter.showBalance
implements com.asera.training.virtualstep.creditDebitTemplate.iflow.behavi
Block
Display Filter 'com.asera.training.virtualstep.creditCardAccount.iflow.displayfilter.showBalance': Missing element 'showCreditCardAccount' from Behavior Filt
Interactive Step 'com.asera.training.virtualstep.creditDebitTemplate.iflow.busstep.showBalancePage': Empty 'Default' Property
Object validation: 2 errors and 1 warnings.

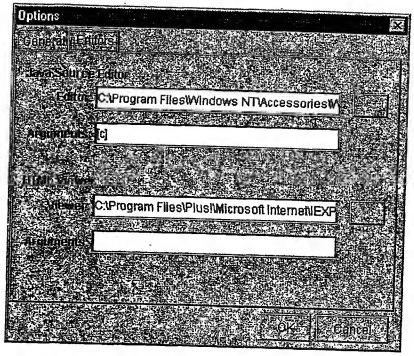

Options
General Editors
Java Source Editor
Editor:
C:\Program Files\Windows NT\Accessories\W
Arguments:
[c]
HTML Viewer
Viewer:
C:\Program Files\Plus!\Microsoft Internet\IEXP
Arguments:
OK
Cancel

Fig. 17

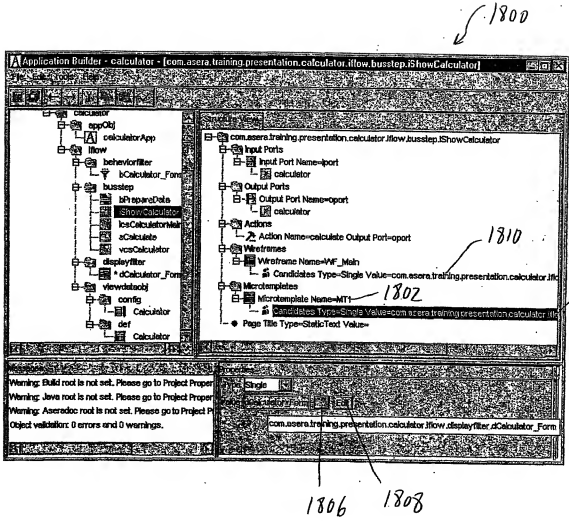

1800
Application Builder - calculator - [com.asera.training.presentation.calculator.iflow.busstep.IShowCalculator]
com.asera.training.presentation.calculator.iflow.busstep.IShowCalculator
Input Ports
Input Port Name=iport
calculator
Output Ports
Output Port Name=oport
calculator
Actions
Action Name=calculate Output Port=oport
Wireframes
Wireframe Name=WF_Main
Candidates Type=Single Value=com.asera.training.presentation.calculator.ifc
Microtemplates
Microtemplate Name=MT1
Candidates Type=Single Value=com.asera.training.presentation.calculator.ifc
Page Title Type=StaticText Value=
1810
1804
1802
1806
1808

Fig. 18

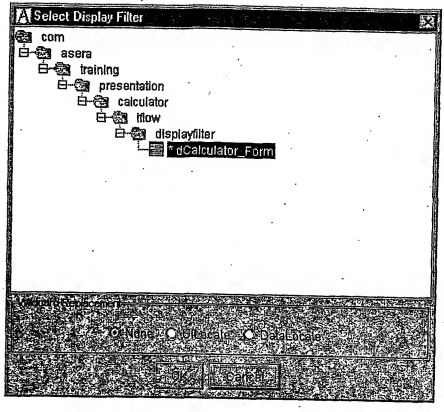

Select Display Filter
com
asera
training
presentation
calculator
iflow
displayfilter
* dCalculator_Form
Cancel

Fig. 19